The generated scanner

The output of `flex` is the file `lex.yy.c', which contains the scanning routine `yylex()', a number of tables used by it for matching tokens, and a number of auxiliary routines and macros. By default, `yylex()' is declared as follows:

```
int yylex()
    {
    ... various definitions and the actions in here ...
    }
```

(If your environment supports function prototypes, then it will be "int yylex(void)".) This definition may be changed by defining the "YY_DECL" macro. For example, you could use:

```
#define YY_DECL float lexscan( a, b ) float a, b;
```

to give the scanning routine the name `lexscan`, returning a float, and taking two floats as arguments. Note that if you give arguments to the scanning routine using a K&R-style/non-prototyped function declaration, you must terminate the definition with a semi-colon (`;').

Whenever `yylex()' is called, it scans tokens from the global input file `yyin` (which defaults to stdin). It continues until it either reaches an end-of-file (at which point it returns the value 0) or one of its actions executes a `return` statement.

If the scanner reaches an end-of-file, subsequent calls are undefined unless either `yyin` is pointed at a new input file (in which case scanning continues from that file), or `yyrestart()' is called. `yyrestart()' takes one argument, a `FILE *' pointer (which can be nil, if you've set up `YY_INPUT` to scan from a source other than `yyin`), and initializes `yyin` for scanning from that file. Essentially there is no difference between just assigning `yyin` to a new input file or using `yyrestart()' to do so; the latter is available for compatibility with previous versions of `flex`, and because it can be used to switch input files in the middle of scanning. It can also be used to throw away the current input buffer, by calling it with an argument of `yyin`; but better is to use `YY_FLUSH_BUFFER` (see above). Note that `yyrestart()' does *not* reset the start condition to `INITIAL` (see Start Conditions, below).

If `yylex()' stops scanning due to executing a `return` statement in one of the actions, the scanner may then be called again and it will resume scanning where it left off.

By default (and for purposes of efficiency), the scanner uses block-reads rather than simple `getc()' calls to read characters from `yyin`. The nature of how it gets its input can be controlled by defining the `YY_INPUT` macro. YY_INPUT's calling sequence is "YY_INPUT(buf,result,max_size)". Its action is to place up to *max_size* characters in the character array *buf* and return in the integer variable *result* either the number of characters read or the constant YY_NULL (0 on Unix systems) to indicate EOF. The default YY_INPUT reads from the global file-pointer "yyin".

A sample definition of YY_INPUT (in the definitions section of the input file):

```
%{
#define YY_INPUT(buf,result,max_size) \
    { \
    int c = getchar(); \
    result = (c == EOF) ? YY_NULL : (buf[0] = c, 1); \
    }
%}
```

This definition will change the input processing to occur one character at a time.

When the scanner receives an end-of-file indication from YY_INPUT, it then checks the `yywrap()' function. If `yywrap()' returns false (zero), then it is assumed that the function has gone ahead and set up yyin to point to another input file, and scanning continues. If it returns true (non-zero), then the scanner terminates, returning 0 to its caller. Note that in either case, the start condition remains unchanged; it does *not* revert to INITIAL.

If you do not supply your own version of `yywrap()', then you must either use `%option noyywrap' (in which case the scanner behaves as though `yywrap()' returned 1), or you must link with `-lfl' to obtain the default version of the routine, which always returns 1.

Three routines are available for scanning from in-memory buffers rather than files: `yy_scan_string()', `yy_scan_bytes()', and `yy_scan_buffer()'. See the discussion of them below in the section Multiple Input Buffers.

The scanner writes its `ECHO' output to the yyout global (default, stdout), which may be redefined by the user simply by assigning it to some other FILE pointer.

Go to the first, previous, next, last section, table of contents.